REMARKS

Claims 1, 6, 8, 14, 19, 21, 27, 28, 29, 30, 31, 33, 34, and 35 have been amended. Claims 1-35 remain pending in the application. Reconsideration is respectfully requested in light of the following remarks.

Claim Objections:

The Examiner objected to claims 1 and 14 for the use of the term "a predicted target address" as this term appears in the amended sections after another "a predicted target address" has been established. Applicants have amended claims 1 and 14 for clarification.

The Examiner objected to claims 27 and 35, as the final limitation does not appear to limit the claim. These claims, as well as claims 28 and 29, have been amended to more clearly describe the claimed invention.

Section 102(b) Rejection:

The Examiner rejected claims 1, 2, 11-15 and 24-26 under 35 U.S.C. § 102(b) as being anticipated by Rotenberg, et al. (hereinafter "Rotenberg"). Applicants respectfully traverse this rejection for at least the following reasons.

Regarding claim 1, contrary to the Examiner's assertion, Rotenburg does not disclose *wherein the prefetch unit is configured to check the trace cache for a match for the predicted target address in response to the branch prediction unit outputting a predicted target address*; and *wherein the prefetch unit is configured to not check the trace cache for a match until the branch prediction unit outputs the predicted target address*. In the Response to Arguments section of the Office Action mailed February 6, 2007, the Examiner submits, "It is inherent that you cannot check for a value if you do not know what the value is. The trace cache as disclosed by Rotenberg cannot search for

the predicted target address in the cache if it does not know what the predicted target address is, and in fact, no cache or any other type of hardware can find something when it doesn't know what it is looking for, thus, the value must have been output from the branch prediction unit prior to the checking."

Applicants assert that the Examiner's interpretation of the operation of Rotenberg is incorrect and that he is also misreading the specific limitations recited in Applicants' claim. For example, in the system of Rotenberg, the trace cache is checked for a match for every fetch address, not only for target addresses generated by the branch prediction unit and BTB logic. As stated on p. 28, second column, beginning of the first full paragraph, "The trace cache is accessed in parallel with the instruction cache and BTB using the current fetch address… The fetch address is used together with the multiple branch predictions to determine if the trace read from the trace cache matches the predicted sequence of basic blocks. Specifically, a trace cache hit requires that (1) the fetch address matches the tag and (2) the branch predictions match the branch flags… On a trace cache hit, an entire trace of instructions is fed into the instruction latch, bypassing the instruction cache" (emphasis added.) The next paragraph goes on to state, "On a trace cache miss, fetching proceeds normally from the instruction cache, i.e., contiguous instruction fetching." It is clear from these passages that every fetch address is compared to the entries in the trace cache, regardless of the operation of the branch prediction unit.

As described in Applicants' Response of October 23, 2006, Rotenberg's trace cache operation is clearly different from Applicants' claimed invention, in that it teaches searching for a hit in the trace cache first (i.e., on every cycle), and then fetching from the instruction cache if there is not a hit. Applicants' claim 1 recites *wherein the prefetch unit is configured to not check the trace cache for a match until the branch prediction unit outputs the predicted target address*. In other words, the trace cache is not checked for a match on every cycle. Instead, it is only checked for a match on cycles for which the branch prediction unit outputs a predicted target address. Therefore, Applicants again assert that Rotenberg clearly does not anticipate claim 1.

For at least the reasons above, the rejection of claim 1 is unsupported by the cited art and removal thereof is respectfully requested. Claim 14 includes limitations similar to claim 1, and so the arguments presented above apply with equal force to this claim, as well.

Section 103(a) Rejection:

The Examiner rejected claims 3 and 16 under 35 U.S.C. § 103(a) as being unpatentable over Rotenberg in view of Patterson, et al. (hereinafter "Patterson"), claims 4, 10, 17 and 23 as being unpatentable over Rotenberg in view of Braught, and further in view of Xia, claims 5-8 and 18-21 as being unpatentable over Rotenberg, Xia, Braught and further in view of Lange, et al. (U.S. Patent 3,896,419) (hereinafter "Lange"), claims 9 and 22 as being unpatentable over Rotenberg, Xia and Braught in view of Akkary, et al. (U.S. Patent 6,247,121) (hereinafter "Akkary"), claims 27-31 and 35 as being unpatentable over Rotenberg, Xia, Braught and Lange in view of Akkary, and claims 32 and 34 as being unpatentable over Rotenberg in view of Xia. Applicants respectfully traverse this rejection for at least the following reasons.

Regarding claim 27, contrary to the Examiner's assertion, the cited references fail to teach or suggest all the limitations of claim 27. For example, The Examiner admits that Rotenberg fails to teach *a method... comprising starting construction of a new trace if the received instruction is associated with a branch label*, as Rotenberg instead teaches starting a trace on a trace cache miss. The Examiner relies on Xia to teach this limitation in Section 5.5, first scheme. This passage describes that trace lines can only start from predictable branches, which does not include all types of branches. It also says nothing about these instructions being associated with a branch label, as required by Applicants' claim. The Examiner cites Braught as teaching, "in Assembly Language, most branches operate on labels, which the machine can only interpret when converted to an address (Page 3). In Braught's example on page 3, all branches are labels, making every branch a label boundary." The Examiner has misquoted Braught, which actually states, "Most of the remaining Assembly Language Instructions operate on Labels!" and does not specify

that most (or all) branches operate on labels, as the Examiner suggests. Therefore, the Examiner's assertion, "When combined with Rotenberg's invention, this would mean that a trace would be generated with an instruction corresponding to a label boundary, as all the branches would go to labels" in unsupported by the cited art.

Furthermore, as discussed at length in Applicants' Response of October 23, 2006, the cited references fails to teach or suggest *receiving a retired instruction*. The Examiner admits that Rotenberg does not teach that instructions need to be retired before the trace can be generated and relies on Akkary to teach this limitation. The Examiner submits that Akkary teaches that instructions are not put into the trace buffers until they have been retired, to ensure that they executed correctly (column 3, lines 40-44). This passage includes the following description:

> Final retirement logic 134 finally retires instructions in trace buffers 114 after it is assured that the instructions were correctly executed either originally or in re-execution.

As noted in Applicants' previous Response, the Examiner has incorrectly interpreted this passage as teaching that instructions are not put into the trace buffers until they have been retired. **This passage actually says just the opposite**, that is, that instructions placed in the trace buffer stay in the trace buffer until they are retired. This is clear from the following two passages (emphasis added):

column 6, lines 53-56: "all needed details regarding the instruction are maintained in trace buffers 114 and MOB 178 until a final retirement, described below."

column 10, lines 1-10: "In one embodiment of the invention, final retirement includes… (3) deallocation of trace buffer and MOB 178 resource entries."

The Examiner previously argued, "Therefore, Akkary suggests the limitation of using a retired instruction, by providing one of ordinary skill in the art with a motivation for using a retired instruction, as opposed to in-flight instructions." In the Response to Arguments section of the present Office Action, the Examiner disagrees with Applicants' argument that Akkary does not teach using a retired instruction for any particular purpose

(much less for building traces) and that the motivation of "incorrect" traces is not valid. The Examiner again includes remarks regarding the correctness of instruction execution, and submits, "The idea behind Examiners statement of an "incorrect" trace is a trace which does not represent the flow of the instruction stream, that is, it does not represent the series of branches that occurred in correct execution of the program, but rather, an alternate series of instructions that did not occur. While this data may be valuable in a trace cache, as it will provide the correct path in the event that the path will be a correct execution in the future, it does not represent what did happen, and Rotenberg's motivation for creating the trace cache was to exploit code reuse, both points being given in Section 1.1, that an instruction which was used recently will be used again in the future, and that branches tend to be biased in one directions, and it is likely that certain paths will be followed frequently." While these goals for Rotenberg's invention are present, this section also explicitly states that the trace line is filled as instructions are fetched from the instruction cache, clearly teaching away from filling the trace line with retired instructions.

The Examiner also points to Rotenberg, Section 2.3, which describes that some traces are committed but never used, thus evicting a useful trace. However, this passage (point 6 "judicious trace selection") describes that a way to avoid this situation is to include an additional small buffer to store recent traces and that the traces in this buffer may only be committed to the trace cache after one or more hits to the trace. This clearly does not suggest waiting until an instruction was retired before considering adding it to the trace, as it refers to a way to avoid a completely unused trace, not an un-retired individual instruction. Similarly, the Examiner's additional reference to point 5 of Section 2.3 ("fill issues") refers to whether or not to fill the trace cache with speculative traces (i.e., constructed traces), not to whether to add an individual un-retired instruction to a trace or to start construction of a trace using an un-retired instruction.

Applicants further assert that the cited references fail to teach or suggest *in response to determining that a previous trace under construction duplicates a trace in a*

trace cache and that the received instruction corresponds to a branch label, beginning construction of a new trace.

The Examiner admits that Rotenberg fails to teach *if a previous trace under construction duplicates a trace in a trace cache, delaying construction of the new trace until the received instruction corresponds to a branch label.* The Examiner admits that Rotenberg does not discuss the issue of duplication, but discusses the disadvantages which occur when a trace cache miss occurs while servicing a previous trace cache miss, and teaches the disadvantages of a useless trace displacing a useful trace. The Examiner then submit, "Therefore, Rotenberg teaches a system in which allows tracing of potential duplicate traces, and also a system which requires action when a miss occurs while servicing a miss." Applicants assert that the Examiner is contradicting himself and that he has cited nothing in Rotenberg that teaches tracing of potential duplicate traces. Applicants again note that, in the system of Rotenberg, the fill-line buffer logic services trace cache misses (Section 2.2, fourth paragraph). That is, the fill-line buffer fetches instructions because the combination of a matching target address and matching branch flags is not found in the trace cache. Therefore, there is no reason to believe there would ever be a duplicate trace in the system of Rotenberg. The performance problem described in Section 2.3, point 5, therefore, would not be solved by checking for duplicate data before building a trace. Since duplicate traces should not be contained in the trace cache of Rotenberg, there would be no opportunity to avoid such duplication.

The Examiner suggests that there is a "potential for duplicate traces to exist with path associativity in Rotenberg's alternative embodiments." **This is completely unsupported in Rotenberg. Similarly, the Examiner's contention that "Rotenberg indicates in his judicial trace selection that storing a duplicate trace would be at best useless, and at worst displace a useful trace that may be used" is also completely unsupported. No such mention of duplicate traces exists in Rotenberg.**

The Examiner submits that Lange teaches, "a system in which a cache is checked for a value while accessing memory, and if the identical value is found in the cache, the

fetch to memory is aborted, freeing the memory to be used by a different operation." Applicants again assert that this description of the operation of a data cache during fetching from main memory has absolutely nothing to do with the limitations of the present invention, as discussed in Applicants' previous Response. A method for reducing fetches from memory by first checking a cache is not analogous to a method for reducing the amount of data that needs to be loaded into a trace cache by checking the contents of the trace cache itself. Finally, even if the problems were analogous, checking Rotenberg's cache before collecting a new trace would not solve the problem described in Section 2.3, point 5, since there should be no duplicate traces generated in Rotenberg and, therefore, no opportunity to "end an unnecessary operation," as previously suggested by the Examiner. Therefore, the Examiner's reasons for combining the references are clearly not supported by the cited art.

For at least the reasons above, the rejection of claim 27 is unsupported by the cited art and removal thereof is respectfully requested. Claim 35 includes limitations similar to claim 27, and so the arguments presented above apply with equal force to this claim, as well.

Regarding claim 32, contrary to the Examiner's assertion, the cited references fail to teach or suggest *a method, comprising: fetching instructions from an instruction cache; continuing to fetch instructions from the instruction cache without searching a trace cache until a branch target address is generated; in response to a branch target address being generated, searching a trace cache for an entry corresponding to the branch target address*. The Examiner submits that Rotenberg teaches all of the limitations of this claim except, "without searching a trace cache". However, as discussed above regarding claims 1 and 14, Rotenberg does not teach searching a trace cache for an entry corresponding to a branch target address in response to a branch target address being generated. In fact, the Examiner admits that Rotenberg teaches that the trace cache is searched on every instruction. The Examiner submits that Xia teaches an advantage of starting a trace only at predictable branch instructions so that less cache space is required. The Examiner submits, "An effect of only having traces start on

branches is that it is then inherent that a non-branch instruction can never be the start of the trace, therefore there would be no reason to search the trace cache for a non-branch instruction, which has further potential advantages such as power saving and potential critical path reduction, as one of ordinary skill in the art would recognize."

Applicants note, however, that in the system taught by Xia, the trace table and instruction cache are checked in parallel on every instruction, just as in the system of Rotenberg. (See Xia, section 3.5, first paragraph.) **Therefore, the Examiner's contention that it would be obvious not to search the trace cache for a non-branch instruction is completely unsupported, as his own references check the trace cache on every instruction. Furthermore, his assertion that such a technique would contribute to power saving or potential critical path reduction is similarly unsupported in the cited art.** In fact, the only advantage noted for starting traces on branches is to save memory space. It is clear from the system of Xia that this memory space can be saved without any change to the method for searching the trace cache on every cycle. Therefore, there clearly is nothing inherent about such a change in a system that starts traces on branches.

For at least the reasons above, the rejection of claim 32 is unsupported by the cited art, and removal thereof is respectfully requested.

In regard to the rejections under both § 102(b) and § 103(a), Applicants assert that numerous ones of the dependent claims recite further distinctions over the cited art. Applicants traverse the rejection of these claims for at least the reasons given above in regard to the claims from which they depend. However, since the rejections have been shown to be unsupported for the independent claims, a further discussion of the dependent claims is not necessary at this time. Applicants reserve the right to present additional arguments.

CONCLUSION

Applicants submit the application is in condition for allowance, and an early notice to that effect is requested.

If any fees are due, the Commissioner is authorized to charge said fees to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/5500-88700/RCK.

Respectfully submitted,

/Robert C. Kowert/
Robert C. Kowert, Reg. #39,255
Attorney for Applicant(s)

Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8850

Date: May 7, 2007